PGPUB-DOCUMENT-NUMBER: 20030154234

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20030154234 A1

TITLE: METHOD FOR TIME PARTITIONED APPLICATION SCHEDULING IN A COMPUTER OPERATING SYSTEM

PUBLICATION-DATE: August 14, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
LARSON, AARON RAYMOND	SHOREVIEW	MN	US

US-CL-CURRENT: 709/107

ABSTRACT:

A time-partitioned system for accounting for processor time consumed by operating system services provided on behalf of an application running in a real-time environment. The time utilized by the operating system is treated as being application processing time, rather than viewing the resultant processor time consumed as an operating system overhead. Each application consists of one or more threads. A time budget is assigned to each thread and to each interrupt. The processor time consumed by the operating system when executing on behalf of a thread or interrupt is charged back to the application or interrupt as part of the application's time budget. Operating system overheads and processor interrupts are thus accounted for accurately enough to determine a schedule for the applications which is enforceable by the operating system.

---------- KWIC ---------

Detail Description Paragraph - DETX (7):

[0034] The method of the present invention implements Rate Monotonic Scheduling (`RMS`) with a **priority inheritance** protocol. The RMS method prioritizes periodic thread execution according to the thread's period. Higher rate threads have higher priority. The **priority inheritance** protocol ensures that threads wishing to lock a mutex are not prevented from doing so by threads having a **lower** priority.

PGPUB-DOCUMENT-NUMBER: 20030014463

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20030014463 A1

TITLE: Task management system

PUBLICATION-DATE: January 16, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
Togawa, Atsushi	Tokyo		JP

US-CL-CURRENT: 709/100

ABSTRACT:

A task management system can inherit priority and can reduce the queue operation required for transition to/return from a mutual exclusion awaiting state in order to reduce an overhead, without operating a dispatch queue. The system can execute a task without considering its priority. The system can start/stop a server task and can inherit priority without operating the dispatch queue. The system includes activity retaining information on task scheduling, context retaining information which is not recorded in the activity, and a dispatch queue used to select the highest priority task from executable tasks. Information on a task is divided and managed by the activity and the context, and each activity is used as one to be inserted into/deleted from the dispatch queue. When the priority of a task is inherited by another task, only the correspondence between activity and context is changed. The system further includes a server task for processing a service request, and a client task for sending a service request to the server task. When the server and client tasks are started or stopped, or when the priority of a task is inherited by another task, only the correspondence between activity and context is changed.

---------- KWIC ---------

Summary of Invention Paragraph - BSTX (8):

[0007] Second, it is possible to minimize the time for which a higher priority task is blocked by a **lower** priority task. By employing a **priority inheritance** protocol (reference: Lui, Sha, Raguntathan Rajkumar, and John P. Lehoczky, "**Priority Inheritance** Protocols: An Approach to Real-time Synchronization", IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185, September 1990), the time for which the higher priority task is blocked can be reduced by using the **lower** priority task. Here, **priority inheritance** protocol is used as a generic term for a basic **priority inheritance protocol and a priority** ceiling protocol.

PGPUB-DOCUMENT-NUMBER: 20020198925

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20020198925 A1

TITLE: System and method for robust time partitioning of tasks in a real-time computing environment

PUBLICATION-DATE: December 26, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
Smith, Joe	Phoenix	AZ	US
Larson, Aaron	Shoreview	MN	US

US-CL-CURRENT: 709/104

ABSTRACT:

A time-partitioned system for accounting for processor time consumed by operating system services provided on behalf of an application runs in a real-time environment. The time utilized by the operating system is treated as application processing time, rather than viewing the resultant processor time consumed as an operating system overhead. Each application consists of one or more threads. A time budget is assigned to each thread and to each interrupt. The processor time consumed by the operating system when executing on behalf of a thread or interrupt is charged back to the application or interrupt as part of the application's time budget. Operating system overheads and processor interrupts are thus accounted for accurately enough to determine a schedule for the applications that is enforceable by the operating system at run time.

---------- KWIC ---------

Detail Description Paragraph - DETX (7):

[0033] The method of the present invention implements Rate Monotonic Scheduling (`RMS`) with a **priority inheritance** protocol. The RMS method prioritizes periodic thread execution according to the thread's period. Higher rate threads have higher priority. The **priority inheritance** protocol ensures that threads wishing to lock a mutex are not prevented from doing so by threads having a **lower** priority.

PGPUB-DOCUMENT-NUMBER:	20020178208			
PGPUB-FILING-TYPE:	new			
DOCUMENT-IDENTIFIER:	US 20020178208 A1			
TITLE:	Priority inversion in computer system supporting multiple processes			
PUBLICATION-DATE:	November 28, 2002			

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Hutchison, Gordon Douglas	Eastleigh		GB	
Peacock, Brian David	North Baddesley		GB	
Trotter, Martin John	Ampfield		GB	

US-CL-CURRENT: 709/102, 709/103 , 709/104

ABSTRACT:

The invention relates to a method of operating a computer system supporting multiple processes having potentially different priorities. The system provides a wait-notify mechanism, whereby a first process can be suspended pending notification from a second process. The mechanism is controlled via a predetermined resource which-must be-owned by the first process when suspension is initiated, and the second process at the time of notification. During the suspension of the first process, the priority of a process that acquires ownership of said predetermined resource is increased, typically to a level equal to that of the first process. This ensures that the first process does not wait an unduly long time to be notified for resumption.

---------- KWIC ---------

Detail Description Paragraph - DETX (11):
[0045] As explained above, the above priority inversion problem has been recognised and addressed in the prior art by **priority inheritance** (or variations thereof). In this known approach, if a thread (T2) is waiting to obtain a monitor (operating as a mutex), then the (high) priority level of thread T2 can be associated with the mutex. This priority is then transferred to the process that currently owns the mutex (in this case T1). Thus T1 essentially receives a temporary priority boost, to allow it to finish its operations with the mutex. Once these operations have been completed, T1 releases the mutex, and drops back down to its original (**lower**) priority. It will be appreciated that this approach is effective at overcoming the priority inversion problem described above, since T1 in its boosted state will take precedence over medium priority thread T3, thereby allowing T1 to quickly release the mutex, and T2 to continue processing.

Detail Description Paragraph - DETX (23):
[0057] To understand how **priority inheritance** operates with respect to the wait set, it will be appreciated that in order to progress the waiting thread (assumed to be high priority), it must be notified, and the thread that notifies it (assumed to have a **lower** priority) must own the monitor at the time of notification. There are two main scenarios to consider:

PGPUB-DOCUMENT-NUMBER: 20020143847

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20020143847 A1

TITLE: Method of mixed workload high performance scheduling

PUBLICATION-DATE: October 3, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
Smith, Gary Stephen	Auburn	CA	US

US-CL-CURRENT: 709/103, 710/40

ABSTRACT:

A method of scheduling in a mixed workload environment. A high priority workload requiring bounded response times is executed on the same system with a low priority workload that is capable of tying up the CPU and multiple volume storage resources of the system by causing multiple concurrent I/O operations, thereby increasing the response times of the high priority workload beyond acceptable bounds. The method of scheduling prevents the response times of the high priority workload from increasing beyond the acceptable bounds by deferring the dispatch of processes servicing the current low priority workload for a time that depends on the priority of the low priority work and by not performing concurrent I/O operations that are requested by the current low priority work, when there is sufficient higher priority activity on the storage resource.

---------- KWIC ---------

Summary of Invention Paragraph - BSTX (7):

[0006] To alleviate this problem, a form of **priority inheritance** is used. The server process that services the client requests is allowed to temporarily inherit the priority of the highest priority client request and is inserted, in priority order, on a queue of ready-to-run processes for the server. Preferably, the ready-to-run queue is priority-ordered by insertion sorting requests onto the queue. Upon reaching the head of the priority-sorted, ready-to-run queue, the server process is dispatched onto the CPU for execution, with its priority restored to normal high process priority. If, while the server process is waiting on the read-to-run queue, another client request arrives having a higher priority than the current client request, the server process priority is changed to the priority of the higher priority client request and its position on the ready-to-run queue is adjusted, thereby assuring that the highest priority client request is serviced before the **lower** priority client request. Also, other server processes of equal and higher priority relative to the priority of the client request are serviced by the server CPU before the **lower** priority client request is serviced. Requiring the server process for the client request to wait behind both equal and higher priority processes on the server, ensures that the scheduling policy is fair and the resulting performance is predictable. This is especially important for real-time applications such as transaction processing.

PGPUB-DOCUMENT-NUMBER: 20020138679

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20020138679 A1

TITLE: System and method for priority inheritance

PUBLICATION-DATE: September 26, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
Koning, Maarten	Bloomfield		CA
Gaiarsa, Andrew	Carp		CA

US-CL-CURRENT: 710/244

ABSTRACT:

A method for controlling **priority inheritance** in a computer system is described, the method including testing a **priority inheritance** variable associated with a task, and **lowering** the current priority of a task when testing the **priority inheritance** variable indicates that the task holds no resources that are involved in a **priority inheritance**.

---------- KWIC ---------

Abstract Paragraph - ABTX (1):

A method for controlling **priority inheritance** in a computer system is described, the method including testing a **priority inheritance** variable associated with a task, and **lowering** the current priority of a task when testing the **priority inheritance** variable indicates that the task holds no resources that are involved in a **priority inheritance**.

Summary of Invention Paragraph - BSTX (13):

[0011] In accordance with an example embodiment of the present invention, a method is provided that includes testing a **priority inheritance** variable associated with a task, and **lowering** a current priority of the task when testing the **priority inheritance** variable indicates that the task holds no resources that are involved in a **priority inheritance**.

Summary of Invention Paragraph - BSTX (19):

[0017] Also in accordance with an example embodiment of the present invention, an article of manufacture is provided, the article of manufacture including a computer-readable medium having stored thereon instructions adapted to be executed by a processor, the instructions which, when executed, define a series of steps to be used to control **priority inheritance,** the steps including: testing a **priority inheritance** variable associated with a task, and **lowering** a current priority of the task when testing the **priority inheritance** variable indicates that the task holds no resources that are involved in a **priority inheritance**.

Detail Description Paragraph - DETX (18):

[0049] In step 312, the current priority of the task holding the semaphore may be set to the maximum of the current priority of the task holding the semaphore and the current priority of the task requesting the semaphore. The task's original or "base" priority may be saved so that when all semaphores held by the task that result in **priority inheritance are released, the task's priority** may be **lowered** to its original priority.

Detail Description Paragraph - DETX (47):
[0074] The example task control block may include a variable 702 indicative of normal or "base" priority of the task, i.e., the priority that the task should currently run, assuming the task has not received a higher, inherited priority. The task control block may further include a current priority 704, a variable that indicates the priority the task should run in light of any **priority inheritance** that may have occurred. In the example embodiment, 702 and 704 may be implemented as integer numbers from 0 to some predetermined upper bound (e.g., 255). It will be appreciated that any consistently-used convention for designating task priorities could be used, e.g., 0 could be the highest priority or the lowest priority, although for clarity in this description it is assumed that **lower** numbers imply **lower** priorities.

Detail Description Paragraph - DETX (87):
[0111] It will be appreciated that, if the requesting task's current priority has been reduced since it first requested the semaphore, the priority comparison of the current priority of the requesting task with the base priority of the semaphore holding task may not detect that the semaphore request was involved in a **priority inheritance**. Accordingly, it will be appreciated that other procedures in the system that may result in **lowering** the priority of the requesting task may need to be modified to adjust the **priority inheritance** variables associated with the holding task and the semaphore when the priority of a requesting task is reduced. It will also be appreciated that other conventional mechanisms could be used to track whether a particular request is involved in a **priority inheritance,** e.g., by including a variable in a task's task control block that tracks whether the task's request for a semaphore has resulted in a **priority inheritance**.

Claims Text - CLTX (1):
1. A method comprising: testing a **priority inheritance** variable associated with a task; and **lowering** a current priority of the task when testing the **priority inheritance** variable indicates that the task holds no resources that are involved in a **priority inheritance**.

Claims Text - CLTX (12):
12. A method comprising: raising a current priority of a task to a current priority of a higher priority task when the higher priority task blocks on a resource held by the task; incrementing a **priority inheritance variable when the higher priority** task blocks on the resource held by the task, the **priority inheritance** variable associated with the task and configured to be indicative of the number of resources held by the task that higher priority tasks are waiting to receive; decrementing the **priority inheritance** variable when the task releases the resource that the higher priority task has blocked on; testing the **priority inheritance** variable; and **lowering** the current priority of the task when testing the **priority inheritance** variable indicates that the task holds no resources that are involved in a **priority inheritance**.

Claims Text - CLTX (32):
32. An article of manufacture comprising a computer-readable medium having stored thereon instructions adapted to be executed by a processor, the instructions which, when executed, define a series of steps to be used to

control **priority inheritance,** said steps comprising: testing a **priority inheritance** variable associated with a task; and **lowering** a current priority of the task when testing the **priority inheritance** variable indicates that the task holds no resources that are involved in a **priority inheritance**.

PGPUB-DOCUMENT-NUMBER: 20020052726

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20020052726 A1

TITLE: Performance simulation apparatus, performance simulation method, and recording medium containing performance simulation program

PUBLICATION-DATE: May 2, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
Tajima, Satoshi	Kawasaki-Shi		JP
Igarashi, Masato	Yachiyo-Shi		JP

US-CL-CURRENT: 703/22

ABSTRACT:

According to the present invention, it is possible to easily execute a performance simulation of a system in which a real-time OS is installed. In a performance simulation apparatus for executing a performance simulation of a system, in which a real-time OS is installed, by the use of a software simulation model 2 having a real-time OS model 3 and an application model 4 having at least one task, the real-time OS model 3 has a plurality of components each including at least one function selecting item for specifying the function. The performance simulation apparatus also has a selector 6 configured to select at least part of the function selecting items to specify the functions of the real-time OS model.

---------- KWIC ---------

Detail Description Paragraph - DETX (8):

[0048] The semaphore function includes function selecting items such as Task Deletion Protection, **Priority Inheritance Method, Semaphore Priority** Value, Semaphore Number Initial Value, Queue size, Polling, etc, as shown in FIG. 2. Task Deletion Protection is an item for inhibiting the deletion of a task having obtained a semaphore. **Priority Inheritance** Method is an item relating to a method for temporarily changing the priority level of a task having obtained a semaphore. Semaphore Priority Value is a value of the priority level of semaphore; if the priority level of a task having obtained a semaphore is **lower** than that of the semaphore itself, the priority level of the task is temporarily changed to the same level as the semaphore. Semaphore Number Initial Value is an initial value of a semaphore. Queue Size is an upper limit of the number of tasks that can be inserted into a queue a semaphore has. Polling is a processing method for processing a task having failed to obtain a semaphore.

US-PAT-NO: 6560628

DOCUMENT-IDENTIFIER: US 6560628 B1

TITLE: Apparatus, method, and recording medium for scheduling execution using time slot data

DATE-ISSUED: May 6, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Murata; Seiji	Tokyo	N/A	N/A	JP

US-CL-CURRENT: 709/103, 709/100 , 709/102

ABSTRACT:

A scheduling method for use with a multi-thread system which is capable of time-sharing processing a plurality of threads is provided which can avoid the drawback of priority inversion, minimize the modification of a wait queue, and ensure the optimum use of the processing time of a CPU.

According to the present invention, a time slot data is assigned to each thread and the scheduling is carried out on the basis of the time slot data. As a processing time is imparted to the time slot data, the execution of the thread to which the time slot data is assigned is started. In case that a higher priority thread has to wait for the completion of the execution of a lower priority thread, the time slot data assigned to the higher priority thread is handled as a time slot data of the lower priority thread, hence allowing the execution of the lower priority thread to be started upon a processing time imparted to the time slot data.

12 Claims, 27 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 17

---------- KWIC ---------

Brief Summary Text - BSTX (11):
For eliminating the drawback of **priority inversion, priority inheritance schemes have been introduced (such as "Priority Inheritance** Protocols: An Approach to Real-time Synchronization" by Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky, Technical Report CMU-CS-87-181, the Computer Science Department of Carnegie Mellon University, November 1987). The **priority inheritance** scheme is designed in which when a higher priority thread is turned to its wait state due to the presence of a **lower** priority thread, the **lower** priority thread inherits the priority level of the higher priority thread.

Brief Summary Text - BSTX (12):
FIG. 2 illustrates a scenario, similar to that of FIG. 1, except for a **priority inheritance** scheme is employed. With the **priority inheritance** scheme, when the execution of a thread C is started with a thread A being in the wait state, the priority level of the thread C is changed to a level equal to the priority level of the thread A. As a result, interruption of the execution of

the thread C by the execution of a thread (for example, the thread B) of which the priority level is **lower** than that of the thread A will be prevented before the thread C leaves from a critical section CS1.

Brief Summary Text - BSTX (14):
To implement the above described **priority inheritance** scheme, the following procedure has to be performed. It is noted that this procedure is explained for a case that the execution of the higher priority thread A is interrupted until the **lower** priority thread C explicitly leaves from the critical section CS1.

Brief Summary Text - BSTX (17):
By following the foregoing procedure, the **priority inheritance** scheme shown in FIG. 2 is implemented where the interruption of the execution of the thread C by a thread (for example, the thread B) of which the priority level is **lower** than that of the thread A can be avoided before the thread C leaves out from the critical section CS1.

Brief Summary Text - BSTX (26):
The scheduling apparatus of the present invention allows the time slot data to be designated separately of the executable subject and subjected to the scheduling, whereby when the executable subject of a higher priority level has to wait for the completion of the execution of the executable subject of a **lower priority level, the priority inheritance** can be carried out by implementing transfer of the time slot data which is low in the cost.

Brief Summary Text - BSTX (30):
The scheduling method of the present invention allows the time slot data to be designated separately of the executable subject and subjected to the scheduling, whereby when the executable subject of a higher priority level has to wait for the completion of the execution of the executable subject of a **lower priority level, the priority inheritance** can be carried out by implementing transfer of the time slot data which is low in the cost.

Brief Summary Text - BSTX (36):
Also, according to the present invention, the **priority inheritance** is implemented by handing the time slot data assigned to the higher priority thread as the time slot data of the **lower** priority thread. This resolves frequent modification of the wait queue in the **priority inheritance**. Therefore, the **priority inheritance** can be conducted without proceeding frequent modification of the wait queue which may increase undesirable overhead.

Detailed Description Text - DETX (5):
According to the present invention, the **inheritance of the priority** level is implemented by dynamically modifying the relation between a thread and its time slot data. More particularly, in case that a higher priority thread has to wait for the completion of the execution of a **lower** priority thread, the time slot data representing the higher priority thread is treated as the time slot data of the **lower** priority thread. When a processing time is then imparted to the latter time slot data, the execution of the **lower** priority thread is started.

Detailed Description Text - DETX (104):
In the scheduling method of the present invention, when the higher priority thread has to wait for the completion of the execution of the **lower** priority

thread, such troublesome operations as modifying the wait queue or recalculating the **priority level for priority inheritance are not needed to have the effect that the priority** level is transferred from the higher priority thread to the **lower** priority thread. Accordingly, the scheduling method is advanced to resolve the drawback of priority inversion which may be critical in the real-time system. Also, without modifying the wait queue nor recalculating the **priority level for priority inheritance,** the scheduling method will avoid increase of the cost in eliminating the drawback of priority inversion. Moreover, the scheduling method allows the processing time of the CPU imparted to a thread at its wait state to be assigned to an interrupting thread which causes the wait state of the thread, hence ensuring the optimum use of the processing time of the CPU without loss.

US-PAT-NO: 6560627

DOCUMENT-IDENTIFIER: US 6560627 B1

TITLE: Mutual exclusion at the record level with priority inheritance for embedded systems using one semaphore

DATE-ISSUED: May 6, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
McDonald; Michael F.	San Jose	CA	N/A	N/A
Arora; Sumeet	Milpitas	CA	N/A	N/A
Chu; Mark	Cupertino	CA	N/A	N/A

US-CL-CURRENT: 709/103, 709/100 , 709/104

ABSTRACT:

A method for providing mutual exclusion at a single data element level for use in embedded systems. Entries for tasks that are currently holding a resource are stored in a hold list. Entries for tasks that are not currently executing and are waiting to be freed are stored in a wait list. A single mutual exclusion semaphore flags any request to access a resource.

24 Claims, 7 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 7

---------- KWIC ---------

Detailed Description Text - DETX (20):
As described above, in one embodiment of the present invention, a **priority inheritance** scheme between two or more tasks is implemented. For example, in step 410 of FIG. 4, the current priority level of task B is shifted to that of task A. This **priority inheritance** mechanism is facilitated by the storage of each tasks' current and original priority in the hold list. The **inheritance of a higher priority** level by a **lower** priority task helps to ensure that the second task (the original **lower** priority task) executes quickly, and thus releases the resource quickly. **Priority inheritance** also prevents pre-emption of the second task by a third task. For example, by inheriting task A's priority level, task B will not get preempted by a third task ("task C"), where task C has a **lower** priority than task A but a higher level than task B's original level. Without such **priority inheritance** mechanism, task C could be allowed to run indefinitely without allowing task A to re-lock the resource temporarily held by task B.

Detailed Description Text - DETX (36):
In certain circumstances, it may occur that a referenced resource cannot be locked. FIG. 7 is a flowchart that illustrates the steps of locking multiple resources in which a referenced resource cannot be locked, according to one embodiment of the present invention. In the following discussion, it is

assumed that a first resource ("resource R") cannot be locked. In step 702 the system unlocks all locked resources L in the reference list with **lower** priority than resource R. For each resource L, the system traverses the derived list and unlocks any resource derived from these resources, step 704. The system next unlocks all locked resources in the derived list with **lower** priority than resource R, step 706. In step 708 the current task is placed in the wait list. In step 710 **priority inheritance** among the competing tasks is checked. The system then releases the semaphore and suspends execution, step 712. When resource R becomes available, and the task waiting for it is resumed, the system then re-acquires the semaphore and continues execution, step 714. In most cases, the system will continue execution from step 602 in the process outlined in FIG. 6.

US-PAT-NO: 5937187

DOCUMENT-IDENTIFIER: US 5937187 A

TITLE: Method and apparatus for execution and preemption control of computer process entities

DATE-ISSUED: August 10, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Kosche; Nicolai	San Francisco	CA	N/A	N/A
Singleton; Dave	Cupertino	CA	N/A	N/A
Smaalders; Bart	San Jose	CA	N/A	N/A
Tucker; Andrew	Los Altos	CA	N/A	N/A

US-CL-CURRENT: 709/104, 709/102

ABSTRACT:

In a multiprocessing computer system, a schedulable process entity (such as a UNIX process, a Solaris lightweight process, or a Windows NT thread) sets a memory flag (sc.sub.-- nopreempt) before acquiring a shared resource. This flag tells the operating system that the process entity should not be preempted. When it is time for the process entity to be preempted, but sc.sub.-- nopreempt is set, the operating system sets a flag (sc.sub.-- yield) to tell the process entity that the entity should surrender the CPU when the entity releases the shared resource. However, the entity is not preempted but continues to run. When the entity releases the shared resource, the entity checks the sc.sub.-- yield flag. If the flag is set, the entity makes an OS call to surrender the CPU.

68 Claims, 1 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 1

---------- KWIC ---------

Detailed Description Text - DETX (14):

In some embodiments, preemption control of FIG. 1 helps to solve the priority inversion problem in which a **lower**-priority LWP holds a resource required by a higher priority LWP, thereby blocking that higher-priority LWP. In Solaris, if a locked object (resource) is known to the kernel, this problem is addressed by **priority inheritance**. The Solaris kernel maintains information about locked objects (mutexes, reader/writer locks, etc). The kernel identifies which thread (LWP) 120.i is the current owner of an object and also which thread is blocked waiting to acquire the object. When a high priority thread (LWP) blocks on a resource held by a **lower** priority thread, the kernel temporarily transfers the blocked thread's priority to the **lower**-priority thread. When this holding thread releases the resource, its priority is restored to its **lower** level. See "Multiprocessor System Architectures", cited above, page 227.

US-PAT-NO: 5872909

DOCUMENT-IDENTIFIER: US 5872909 A

TITLE: Logic analyzer for software

DATE-ISSUED: February 16, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Wilner; David N.	Oakland	CA	N/A	N/A
Smith; Colin	Alameda	CA	N/A	N/A
Cohen; Robert D.	Oakland	CA	N/A	N/A
Burd; Dana	Oakland	CA	N/A	N/A
Fogelin; John C.	Berkeley	CA	N/A	N/A
Fox; Mark A.	San Francisco	CA	N/A	N/A
Long; Kent D.	Richmond	CA	N/A	N/A
Burns; Stella M.	San Francisco	CA	N/A	N/A

US-CL-CURRENT: 714/38, 714/47

ABSTRACT:

The present invention logs events which occur in the target software, and stores these in a buffer for periodic uploading to a host computer. Such events include the context switching of particular software tasks, and task status at such context switch times, along with events triggering such a context switch, or other events. The host computer reconstructs the real-time status of the target software from the limited event data uploaded to it. The status information is then displayed in a user-friendly manner. This provides the ability to perform a logic analyzer function on real-time software. A display having multiple rows, with one for each task or interrupt level, is provided. Along a time line, an indicator shows the status of each program, with icons indicating events and any change in status.

30 Claims, 18 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 14

---------- KWIC ---------

Detailed Description Text - DETX (26):

Each task has a "priority" which indicates that task's eligibility to control the CPU relative to the other tasks in the system. A task is in the inherited state when its priority has been increased because it owns a mutual exclusion semaphore that has **priority inheritance enabled and a higher-priority** task is waiting for that semaphore. **Priority inheritance is a solution to the priority** inversion problem: a higher-priority task being forced to wait an indefinite period of time for the completion of a **lower**-priority task. For example, assume that task 30 needs to take the mutual exclusion semaphore for a region, but taskLow currently owns the semaphore. Although taskHi preempts taskLow, taskHi pends immediately because it cannot take the semaphore. Under some circumstances, taskLow can then run, complete its critical region and release the semaphore, making taskHi ready to run. If, however, a third task, taskMed, preempts taskLow such that taskLow is unable to release the semaphore, then taskHi will never get to run. With the **priority inheritance** option enabled, a task, such as taskLow, that owns a resource executes at the priority of the highest-priority task pended on that resource, the priority of taskHi in the example. When the task gives up the resource, it returns to its normal priority.

DERWENT-ACC-NO: 2002-740589

DERWENT-WEEK: 200280

TITLE: **Priority inheritance** control method in multitasking computer operating system, involves testing **priority inheritance** variable associated with task and **lowering** current priority of task, accordingly

INVENTOR: GAIARSA, A; KONING, M

PATENT-ASSIGNEE: GAIARSA A[GAIAI] , KONING M[KONII]

PRIORITY-DATA: 2001US-0812752 (March 20, 2001)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 20020138679 A1	September 26, 2002	N/A	029	G06F 012/00

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
US20020138679A1	N/A	2001US-0812752	March 20, 2001

INT-CL (IPC): G06F012/00, G06F013/14 , G06F013/38

ABSTRACTED-PUB-NO: US20020138679A

BASIC-ABSTRACT:

NOVELTY - A **priority inheritance** variable associated with a task is tested. The current priority of the task is **lowered** when the tested variable indicates that the task does not hold resources that are involved in the **priority inheritance**. The current priority of task is raised, when a higher priority task blocks on a resource held by the task.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are included for the following:

(1) Priority inheritance control system; and

(2) Article of manufacture comprising computer readable medium storing priority inheritance control program.

USE - For controlling priority inheritance in multitasking computer operating system such as Unix and windows.

ADVANTAGE - By lowering the priority of the task, based on the resources held by the task, the resources are allocated effectively among various user applications that are run simultaneously.

DESCRIPTION OF DRAWING(S) - The figure shows the schematic illustration of the flow diagram explaining the execution of tasks without priority inheritance.

CHOSEN-DRAWING: Dwg.1/16

TITLE-TERMS: PRIORITY CONTROL METHOD COMPUTER OPERATE SYSTEM TEST PRIORITY

VARIABLE ASSOCIATE TASK LOWER CURRENT PRIORITY TASK ACCORD

DERWENT-CLASS: T01

EPI-CODES: T01-F02A1; T01-F05G3; T01-S03;

SECONDARY-ACC-NO:
Non-CPI Secondary Accession Numbers: N2002-583543

US-PAT-NO: 6397243

DOCUMENT-IDENTIFIER: US 6397243 B1

TITLE: Method and device for processing several technical applications each provided with its particular security

DATE-ISSUED: May 28, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Colas; Gerard	Versailles	N/A	N/A	FR
Guedou; Philippe	Rambouillet	N/A	N/A	FR
Le Borgne; Olivier	Montigny le Bretonneux	N/A	N/A	FR
Rowenczyn; Jean-Jacques	Ris-Orangis	N/A	N/A	FR

US-CL-CURRENT: 709/100, **710/47** , **710/48** , **710/52** , 712/35 , 712/36

ABSTRACT:

Method of processing several computer-controlled technical applications. The applications are executed within the same computer working in successive work cycles by allotting thereto during the work cycles at least one time slot of a previously fixed duration. At the end of the time slot allotted to a technical application, a start interrupt is generated which is aimed at starting the execution of another technical application. Each technical application has allotted thereto at least one memory space slot for writing data. The memory space slot is write-inaccessible to the other technical applications so that a technical application which during execution possesses a given level of criticality does not disturb another application having a higher or equal level of criticality.

20 Claims, 11 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 7

---------- KWIC ---------

Current US Cross Reference Classification - CCXR (1):
710/47

Current US Cross Reference Classification - CCXR (2):
710/48

Current US Cross Reference Classification - CCXR (3):
710/52

Other Reference Publication - OREF (2):
"**Priority Inheritance** Protocols: An Approach to Real-Time Synchronization",

Lui Shi, Ragunathan Rajkumar and John P. Lehoczky, 1990 IEEE.*

US-PAT-NO: 5515538

DOCUMENT-IDENTIFIER: US 5515538 A
See image for Certificate of Correction

TITLE: Apparatus and method for interrupt handling in a multi-threaded operating system kernel

DATE-ISSUED: May 7, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Kleiman; Steven R.	Los Altos	CA	N/A	N/A

US-CL-CURRENT: **710/260**, 709/103 , 709/108

ABSTRACT:

The disclosed invention is a method and apparatus for use in handling interrupts in a data processing system where the kernel is preemptible, has real-time scheduling ability, and which supports multithreading and tightly-coupled multiprocessors. The invention more specifically provides a technique for servicing interrupts in a processor by means of kernel interrupt handler threads which service the interrupt from start to finish. For efficiency, the interrupt handler threads do not require a complete context switch unless the interrupt handler thread is blocked. The kernel makes use of preprepared interrupt handler threads for additional efficiency, and these interrupt handler threads are not subjected to inordinate delays caused by the phenomenon of interrupt priority inversion if they do become blocked.

20 Claims, 12 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 12

---------- KWIC ---------

Detailed Description Text - DETX (10):
The mutex and writer locks support a dispatching **priority inheritance** protocol which prevents lower priority threads from blocking higher priority threads (priority inversions).

Detailed Description Text - DETX (61):
If the interrupt handler thread (thread B) does block (i.e. become blocked) because of some synchronizing condition, the processing which occurs is shown in FIG. 8. Referring to FIG. 8, the kernel puts the blocked thread B on the sleep queue of the synchronizing object related to the thread that is causing the block; and gives the dispatching priority level of the blocked interrupt handler thread B to the thread causing the block 182. This is called "**priority inheritance**" and insures that the blocking thread will run as soon as possible. Then the kernel transfers control to the dispatcher who sees that its register "t.sub.-- intr" shows that thread A is pinned. As a result, the dispatcher calls "thread.sub.-- unpin" 184. "Thread.sub.-- unpin" completes the context switch of thread A (the "pinned" thread); "Nulls" the "t.sub.-- intr" field of thread B; marks the interrupt level of the blocked interrupt thread B in the

"cpu.sub.-- intr.sub.-- active" field in the CPU's cpu structure, so that the interrupt priority level will be held by the interrupted CPU; and control is transferred to the dispatcher to initiate the highest priority thread. This is usually the blocking thread 186. At this point the blocked interrupt handler thread B awaits the end of the condition that caused it to block. When the synchronizing object controlling the sleep queue containing the sleeping thread B finally awakens thread B, the common interrupt code, which called the handler checks the "t.sub.-- intr" field and finds it equal to "Null" indicating thread B was blocked. 188 Thread B completes its handling of the interrupt and upon completion, and if no other blocks occur, returns (158 in FIG. 7b), and the common interrupt code puts the interrupt handler thread back onto the pre-prepared interrupt handler thread pool for that CPU and returns control to the dispatcher to switch to the highest priority runnable thread available. (176 in FIG. 7b).

Current US Original Classification - CCOR (1):
  710/260

PGPUB-DOCUMENT-NUMBER: 20030061395

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20030061395 A1

TITLE: MESSAGE PASSING USING SHARED MEMORY OF A COMPUTER

PUBLICATION-DATE: March 27, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
KINGSBURY, BRENT A.	BEAVERTON	OR	US
SULMONT, JEAN-MARIE CHRISTIAN	PORTLAND	OR	US
MCKENNEY, PAUL E.	BEAVERTON	OR	US

US-CL-CURRENT: **709/312**

ABSTRACT:

A lock-free mechanism is provided for successfully passing messages between processes even if a process is preempted while sending or receiving a message. Messages are communicated between processes using a mailbox data structure stored in memory shared by the processes, without the use of locks or other mutual exclusion entities that would otherwise limit concurrent delivery and receipt of messages placed in the data structure. The data structure in the illustrative embodiment includes one or more message slots for storing messages placed in the data structure and a number of state variables for inserting messages into and removing messages from the message slots. A process sends or retrieves messages by manipulating the state variables using indivisible atomic operations. This ensures that a process cannot be preempted from message passing until it finishes executing the atomic instruction. The method and mechanism have particular value in distributed shared memory (DSM) and non-uniform memory access (NUMA) machines.

---------- KWIC ---------

Current US Classification, US Primary Class/Subclass - CCPR (1):
709/312

Summary of Invention Paragraph - BSTX (12):
[0010] The acquisition and holding of such mutexes, unfortunately, has several disadvantages. It limits the possible concurrency of message-passing software. It also naturally imposes overhead in the form of operating system calls to obtain and release such mutexes. Even worse, it does not handle preemption well. A first process (or thread) may find itself preempted by a second process (or thread) while holding the mutex that guards the shared memory in which messages are being passed. The mutex remains held while the first process is preempted, preventing other processes from accessing this shared memory. If the second process finds itself needing to acquire the mutex held by the preempted first process, then the second process is blocked until the first process is resumed and run to the point of releasing the mutex. If the two processes are running at different fixed priorities, and the operating system strictly enforces priority in its scheduling decisions, then deadlock can occur if the priority of the second process is higher than the first. To

avoid this deadlock in the use of mutexes to guard shared data, the operating system must implement complex **priority-inheritance** mechanisms in which the first process is temporarily resumed with an elevated priority that allows it to complete its work to the point of releasing the mutex to the second process.

PGPUB-DOCUMENT-NUMBER: 20020133530

PGPUB-FILING-TYPE: new

DOCUMENT-IDENTIFIER: US 20020133530 A1

TITLE: Method for resource control including resource stealing

PUBLICATION-DATE: September 19, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
RULE-47			
Koning, Maarten	Bloomfield		CA

✓

US-CL-CURRENT: **709/102**

ABSTRACT:

A method for resource control including resource stealing is disclosed, the method including assigning a resource to a holding task, receiving a request by a higher priority task to take the resource, the higher priority task having higher priority than the holding task, determining whether the holding task has used the resource since the resource was assigned to the holding task, releasing the resource when the higher priority task requests to take the resource and the holding task has not used the resource since the resource was assigned to the holding task; and assigning the resource to the higher priority task.

---------- KWIC ---------

Current US Classification, US Primary Class/Subclass - CCPR (1):
709/102

Detail Description Paragraph - DETX (4):
[0024] An example embodiment implemented according to the present invention may be included as part of a computer environment, e.g., in a computer operating system. The example embodiment may include a priority control mechanism, a mutual exclusion control mechanism, mechanisms to control **priority inheritance,** as well as other conventional features of a computer operating system.

Detail Description Paragraph - DETX (8):
[0028] The example embodiment may also include "mutual exclusion semaphores". Mutual exclusion semaphores may be used to control access to shared resources. Mutual exclusion semaphores in the example embodiment may include several features that make them more suitable than binary semaphores for controlling access to shared resources where mutually exclusive access is desired. In the example embodiment, a mutual exclusion semaphore may generally only be given by the task that took it. Also in the example embodiment, a mutual exclusion semaphore may not be given during an interrupt service routine, a special procedure used to handle hardware interrupts without context switching. Also in the example embodiment, a mutual exclusion semaphore may not be flushed. Mutual exclusion semaphores may also be "inversion safe", i.e., designed to include a mechanism for **priority inheritance that temporarily increase the priority** of low priority tasks holding mutual exclusion semaphores that higher priority tasks are waiting for.

Detail Description Paragraph - DETX (68):
[0088] It will be appreciated that when changes are made to task **priorities due to priority inheritance** that appropriate adjustments will need to be made to the operating system queues, e.g., entries stored in queues in priority order may need to be re-sorted.

US 20020178208A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2002/0178208 A1**

Hutchison et al. (43) **Pub. Date: Nov. 28, 2002**

(54) **PRIORITY INVERSION IN COMPUTER SYSTEM SUPPORTING MULTIPLE PROCESSES**

(75) Inventors: **Gordon Douglas Hutchison**, Eastleigh (GB); **Brian David Peacock**, North Baddesley (GB); **Martin John Trotter**, Ampfield (GB)

Correspondence Address:
Gregory M. Doudnikoff
IBM Corp, IP Law Dept T81/503
3039 Cornwallis Road
PO Box 12195
Research Triangle Park, NC 27709-2195 (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(21) Appl. No.: **10/155,300**

(22) Filed: **May 24, 2002**

(30) **Foreign Application Priority Data**

May 24, 2001 (GB) 0112571.5

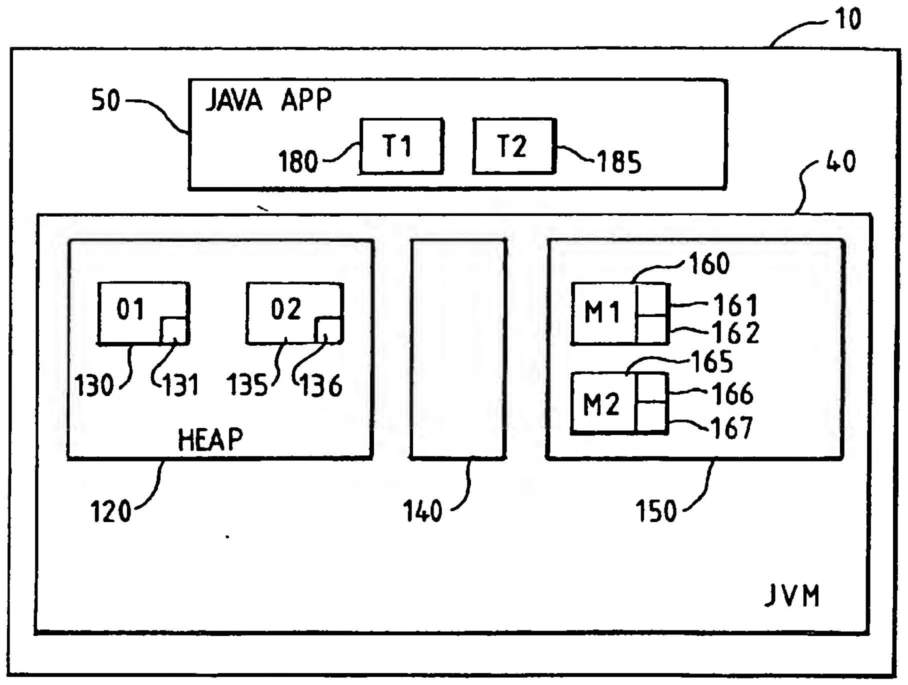
Publication Classification

(51) **Int. Cl.**^{7} **G06F 9/00**

(52) **U.S. Cl.** **709/102**; 709/103; 709/104

(57) **ABSTRACT**

The invention relates to a method of operating a computer system supporting multiple processes having potentially different priorities. The system provides a wait-notify mechanism, whereby a first process can be suspended pending notification from a second process. The mechanism is controlled via a predetermined resource which-must be-owned by the first process when suspension is initiated, and the second process at the time of notification. During the suspension of the first process, the priority of a process that acquires ownership of said predetermined resource is increased, typically to a level equal to that of the first process. This ensures that the first process does not wait an unduly long time to be notified for resumption.


10
50
JAVA APP
180
T1
T2
185
40
01
02
130
131
135
136
HEAP
120
140
M1
160
161
162
M2
165
166
167
150
JVM